Tabla de contenido

[1 Arquitectura J1 1](#_Toc381533799)

[1.1 Generalidades 1](#_Toc381533800)

[1.1.1 Data stack 3](#_Toc381533801)

[1.1.2 Return Stack 3](#_Toc381533802)

[1.1.3 Dual port RAM 4](#_Toc381533803)

[1.1.4 ALU 4](#_Toc381533804)

[1.2 Algoritmo 4](#_Toc381533805)

[1.2.1 Seleccionar operación ALU 5](#_Toc381533806)

[1.2.2 Asignar \_st0 (Ejecutar operación ALU) 5](#_Toc381533807)

[1.2.3 Asignar Program Counter 5](#_Toc381533808)

[1.2.4 Asignar direcciones de stacks 6](#_Toc381533809)

[1.2.5 Reset del sistema 6](#_Toc381533810)

[1.2.6 Escribir en Stacks 7](#_Toc381533811)

[2 Bloques 7](#_Toc381533812)

[2.1 Dual port ram – dpram 7](#_Toc381533813)

[2.2 Uart 8](#_Toc381533814)

[3 Simulaciones de comprobación a partir de la memoria del programa 9](#_Toc381533815)

[4 Ejemplo de programación serie de Fibonacci usando lenguaje GForth 11](#_Toc381533816)

# Arquitectura J1

## Generalidades

J1 es el núcleo de una unidad de procesamiento diseñada para implementar sobre FPGA. Su arquitectura es de 16 bits y contiene básicamente 4 bloques funcionales:

* Unidad de memoria RAM de 16x8K
* Pila de datos de 32 registros
* Pila de respuesta de 32 registros
* Unidad ALU

La unidad RAM contiene, como es común, las instrucciones en lenguaje máquina a ejecutar y los datos que estos manejan. Adicionalmente, contiene espacio reservado para manejo de dispositivos periféricos y de entrada salida. Como característica específica, este núcleo no posee registros de estado ni de banderas. El manejo de bytes debe realizarse a nivel software.

En alto nivel, para ejecutar cada instrucción de un programa se siguen los siguientes pasos:

1. Toma instrucción de memoria RAM (Puerto A) y toma datos de las pilas
2. Decodifica la instrucción
3. Ejecuta operación
4. Guarda resultados en pila de respuesta y en pila de datos. Modifica los registros de datos y manejo de entrada salida en memoria RAM

Estos pasos permiten ver que la obtención de los datos y la decodificación de la operación son tareas paralelas en términos de ejecución lógica. Los puertos de la memoria RAM se usan para distinguir las dos secciones de memoria: el puerto A para acceder a instrucciones/parámetros y el puerto B se encarga del manejo de periféricos. El lenguaje utilizado para esta arquitectura es GForth

***Entradas:***

*Sys\_clk\_i:* Reloj general del sistema. La arquitectura completa es sensible a flancos de subida.

*Sys\_rst\_i:* Reset general del sistema. Con esta señal se inicializan todos los registros internos del núcleo

*Io\_din:* Palabra de 16 bits que sirve para ingresar al sistema. Comúnmente, se utiliza para programar la memoria RAM y para comunicación con periféricos.

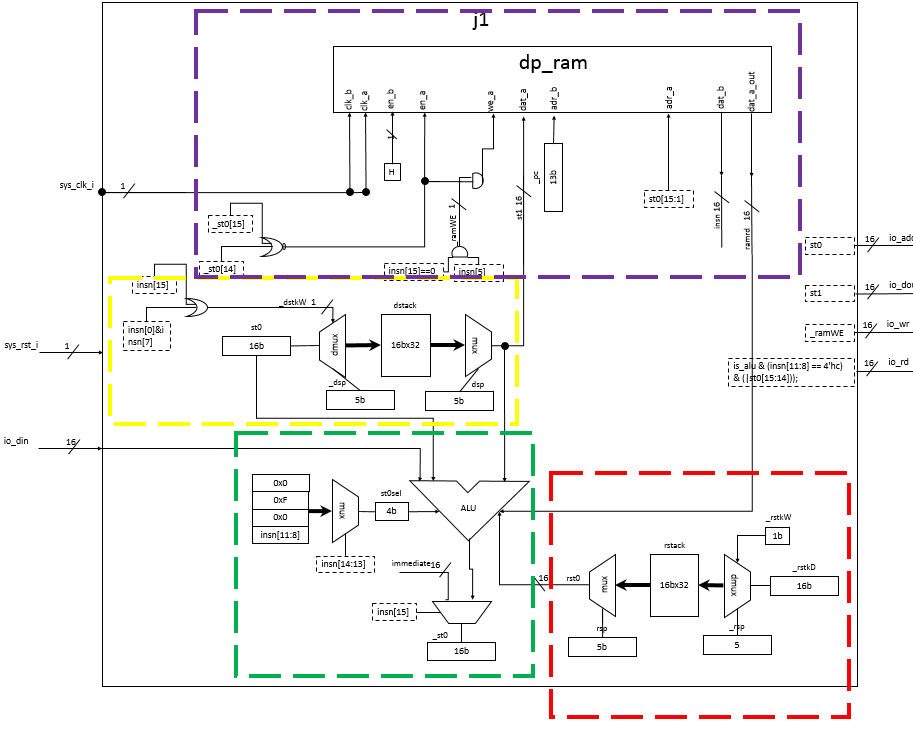
***Salidas:***

*Io\_addr:* Salida del sistema utilizada para registrar las direcciones de memoria de los periféricos. Está asociada al registro st0

*Io\_dout:* Salida del sistema que se utiliza para enviar datos. Está asociada al registro st1.

*Io\_wr:* Salida que informa que se está escribiendo en la memoria RAM

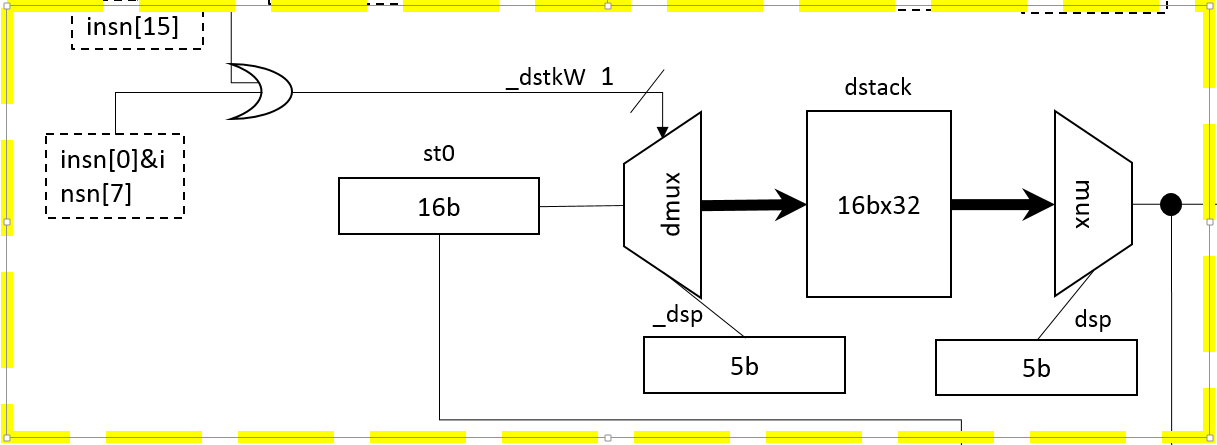
*Io\_rd:* Salida que informa que se están tomando datos de la RAM

En la siguiente figura se pueden identificar los bloques que hacen parte de la arquitectura:

Arquitectura General

En los recuadros, se pueden identificar los 4 bloques funcionales mencionados anteriormente. A continuación se describen de forma general cada uno de ellos:

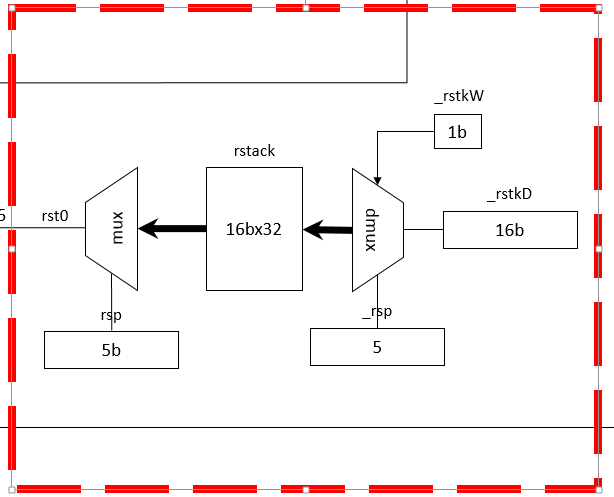
### Data stack



DStack

El bloque *Data Stack* sirve para manejar los datos parciales y de entrada que van hacia la ALU. Tiene secciones de memoria de 16bx32 y es controlado por los registros \_dsp (para escritura) y dsp (para lectura). La habilitación de escritura sobre esta pila depende de la instrucción en ejecución.

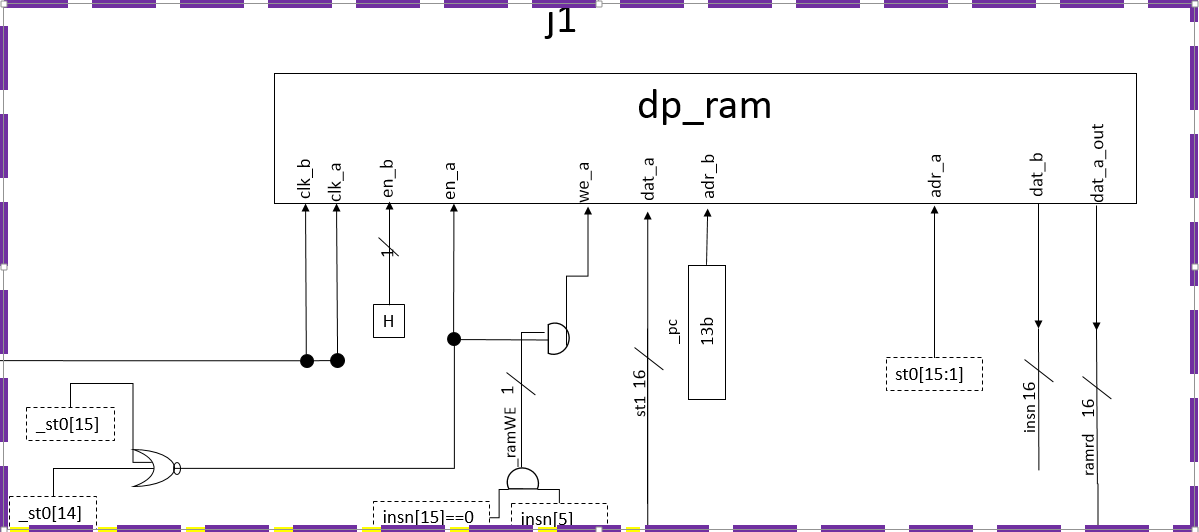
### Return Stack



RStack

Este bloque sirve para almacenar los resultados, definitivos y parciales, de las operaciones que se realizan en la ALU. Es controlado por los registros \_rsp (para escritura), rsp (para lectura) y \_rstlW (como habilitador de escritura). Este tiene adicionalmente un registro de dato, que es modificado según la instrucción y los datos en memoria.

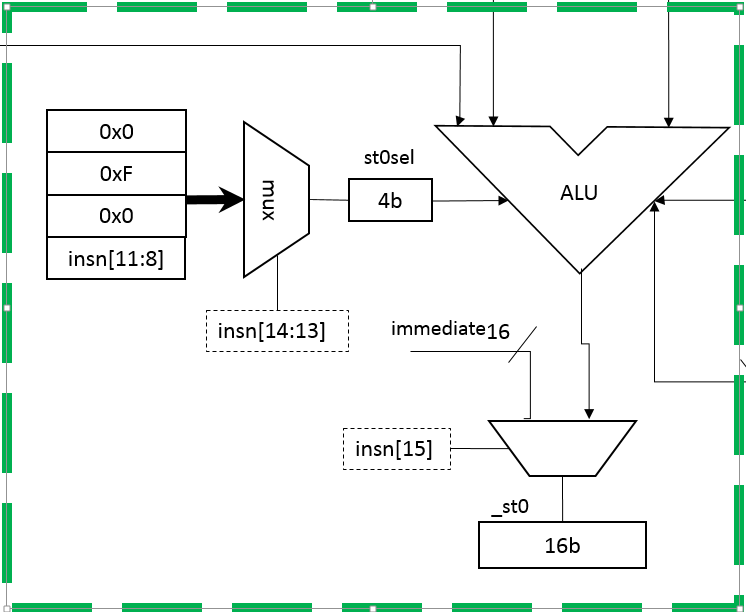
### Dual port RAM



Dual Port RAM

Este bloque tiene por objetivo almacenar la memoria de programa y manejo de periféricos de la arquitectura. Contiene, por tanto, los resultados definitivos de las operaciones y sirve como interfaz para los periféricos que se conecten a la arquitectura.

### ALU

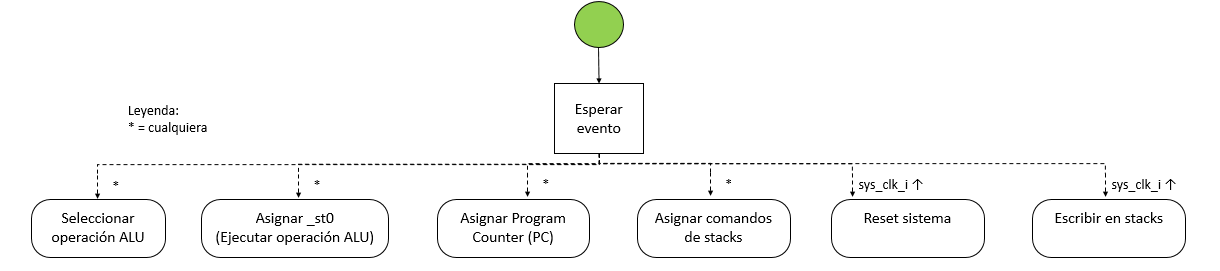


ALU

Este bloque realiza las operaciones atómicas definidas para la arquitectura. Sus entradas son principalmente los registros de las pilas (Rstak, Dstak), pero para ciertas instrucciones se comunica con la memoria RAM. No ejecuta todas las posibles instrucciones de la arquitectura, dado que ésta permite el manejo de memoria, stacks y periféricos externos, pero en general contiene las operaciones básicas. Las operaciones que realiza son mencionadas en el algoritmo general.

## Algoritmo

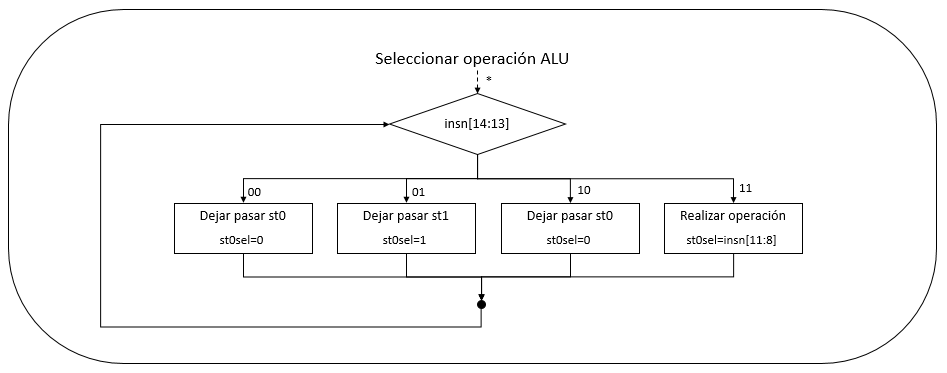
El algoritmo general de la arquitectura incluye 6 procesos que se ejecutan en paralelo. Dos de ellos están sincronizados exclusivamente con el reloj del sistema. Todos los demás son sensibles a cualquier cambio en las entradas. Los procesos ejecutados están relacionados con la ejecución del programa, con el reset del sistema y con la escritura en los stacks.



Macroprocesos de la Arquitectura

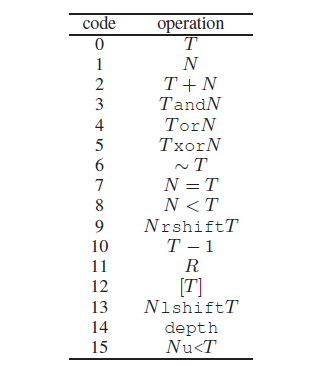
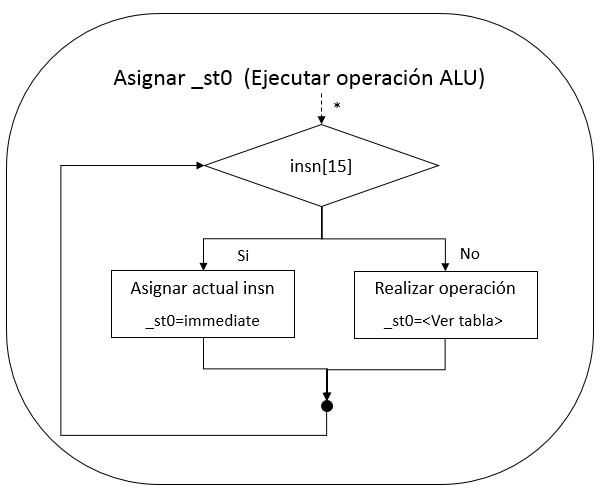
### Seleccionar operación ALU

Este proceso se encarga de codificar la operación que ingresa a la ALU. Esta codificación está dada por el vector de instrucción *insn*. La operación es codificada en la variable st0sel. El siguiente diagrama describe su operación.



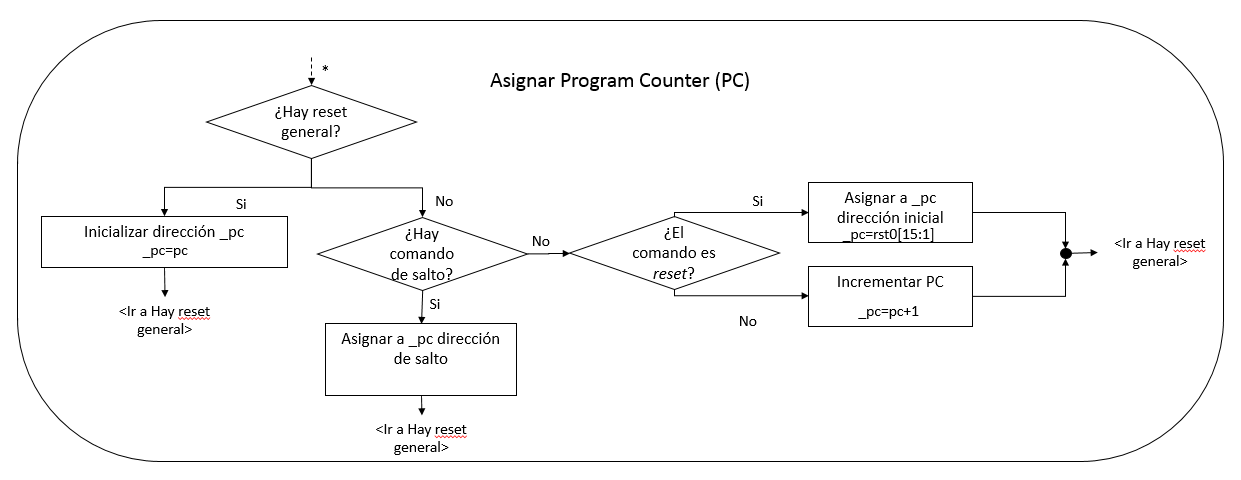
### Asignar \_st0 (Ejecutar operación ALU)

Este proceso se encarga de ejecutar la operación sobre la ALU. Su funcionamiento es activado sólo si la instrucción requiere de la ALU, dado que el registro de resultado (*\_st0*) es el mismo para toda operación de la arquitectura. Las operaciones son mostradas en la figura: T y N representan las entradas estándar de la ALU, que son provenientes del stack de datos (*Dstack*), y R que proviene del stack de resultados (*Rstack*).



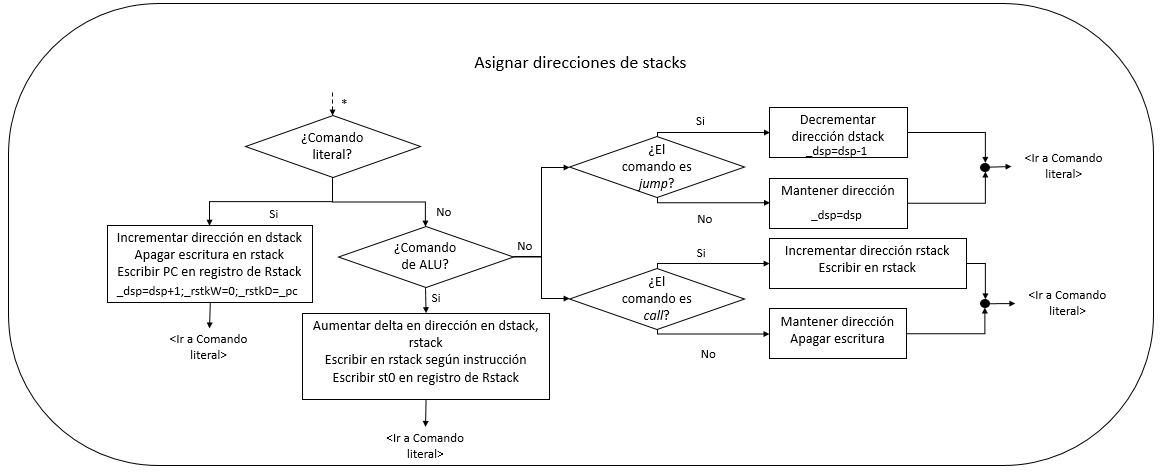
### Asignar Program Counter

Este proceso se encarga de realizar el manejo del Program Counter. Este manejo tiene en cuenta los saltos, los llamados a funciones. El proceso de manejo es similar al de la mayoría de arquitecturas actuales y se muestra en el siguiente diagrama.



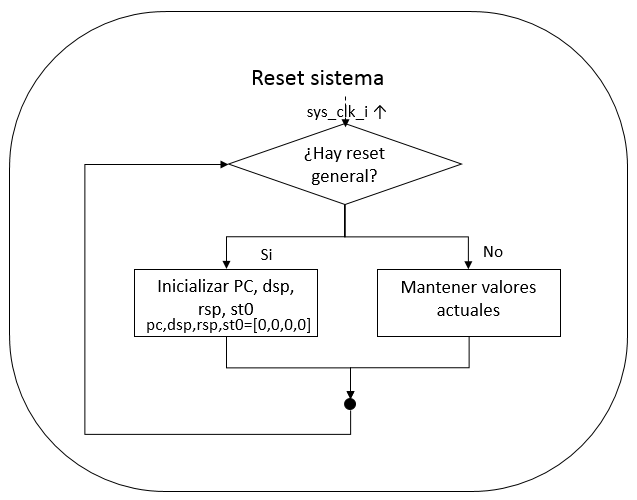
### Asignar direcciones de stacks

El manejo de stacks es uno de los procesos más específicos y necesarios para esta arquitectura. El manejo depende específicamente de la instrucción actual de ejecución. Este proceso sólo asigna los apuntadores a los registros de las pilas. El proceso de escritura En específico el algoritmo utilizado es el siguiente:



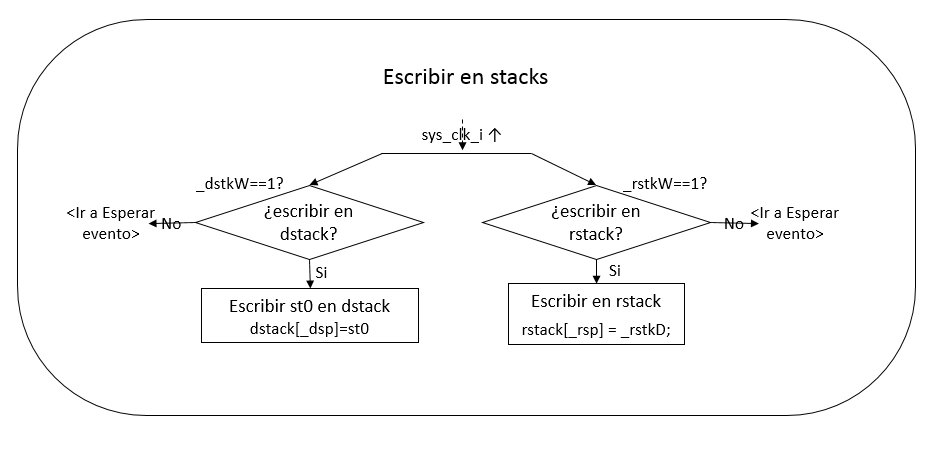
### Reset del sistema

Este proceso ubica en cero todos los apuntadores de la arquitectura.



### Escribir en Stacks

Este proceso se encarga de realizar la escritura sobre las pilas de datos. Utiliza los apuntadores asignados en el proceso de administración y los habilitadores de escritura.



# Bloques

## Dual port ram – dpram

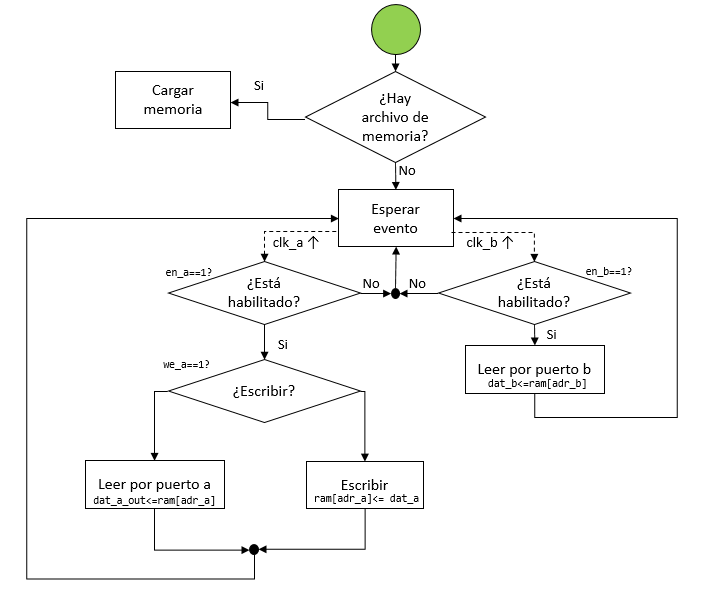
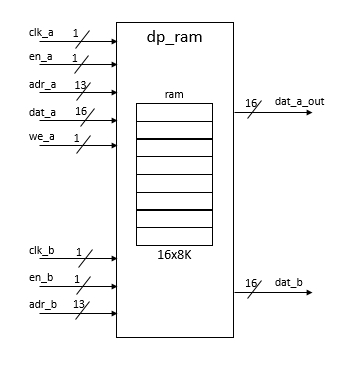


Diagrama de Bloques y Algoritmo de Dpram

Este bloque se encarga de almacenar el código a ejecutar, las variables y parámetros de dicho código y de mantener el estado de registros con los periféricos de la arquitectura. Maneja por un puerto (puerto A) la ejecución del programa y por el otro (puerto B) la información de los periféricos. El puerto A permite lectura y escritura, mientras que el puerto B sólo permite lectura. En la figura se puede visualizar el algoritmo de ejecución de este bloque.

***Entradas:***

*Clk\_a:* Clock para el puerto de lectura y escritura

*En\_a:* Habilita la lectura de la memoria por este puerto

*Adr\_a:* Dirección de memoria en la que se quiere realizar alguna operación

*Dat\_a:* Dato a escribir en la memoria

*We\_a* habilita la escritura de la memoria

*Clk\_b:* Clock para el puerto de lectura

*En\_b:* Habilita la lectura de la memoria por este puerto

*Adr\_b:* Dirección de memoria sobre la que se quiere hacer la lectura de la memoria

***Salidas***

*Dat\_a\_out:* Lectura obtenida por el puerto A

*Dat\_b:* Lectura obtenida por el puerto B

## Uart

Este bloque es uno de los periféricos que pueden ser conectados a la arquitectura J1. Maneja el protocolo serial deforma asíncrona y no contiene todos los puertos del estándar serial. Tiene la posibilidad de configurar la tasa de baudios a la que es enviada la información utilizando dos registros (*dInc, dNxt*). Sólo permite el envío de bits agrupados por Bytes y contiene un indicador de canal ocupado.

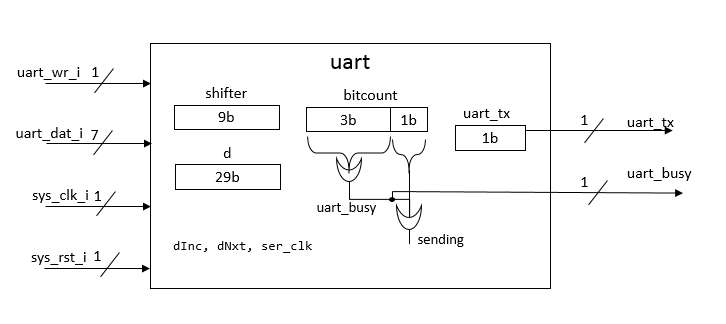
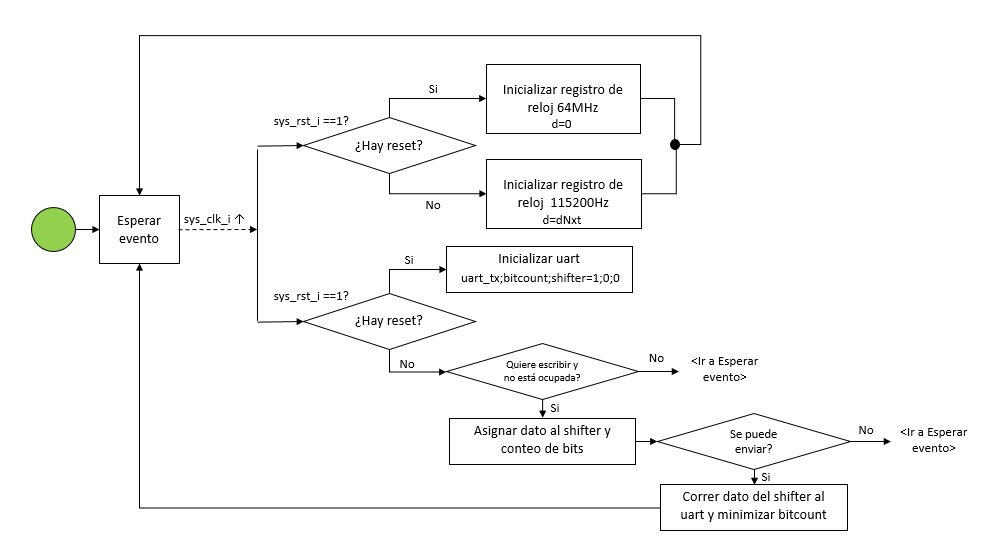


Diagrama de bloques.



Algoritmo de UART

***Entradas:***

*Uart\_wr\_i:* Habilita la escritura del registro que almacena el byte que se quiere enviar

*Uart\_dat\_i:*Dato de 8bits que se quiere enviar por el puerto serial

*Sys\_clk\_i:* Clock general del periférico

*Sys\_rst\_i:* Señal para reset de registros y tasa de baudios.

***Salidas:***

*Uart\_tx:* canal por el que se envían bit a bit los datos del registro

*Uart\_busy:* Señal que indica si se está realizando un envío

# Simulaciones de comprobación a partir de la memoria del programa

El programa que viene por defecto como ejemplo para la arquitectura del J1, usa el periférico de UART para transmitir los caracteres “HELLO” de manera serial usando un shift register para convertir los datos paralelos en una secuencia de datos seriales. La CPU lee desde su memoria RAM las instrucciones que le permiten usar la señal de salida J1\_io\_dout (Fig.1-1) para enviar datos hacia el módulo UART.

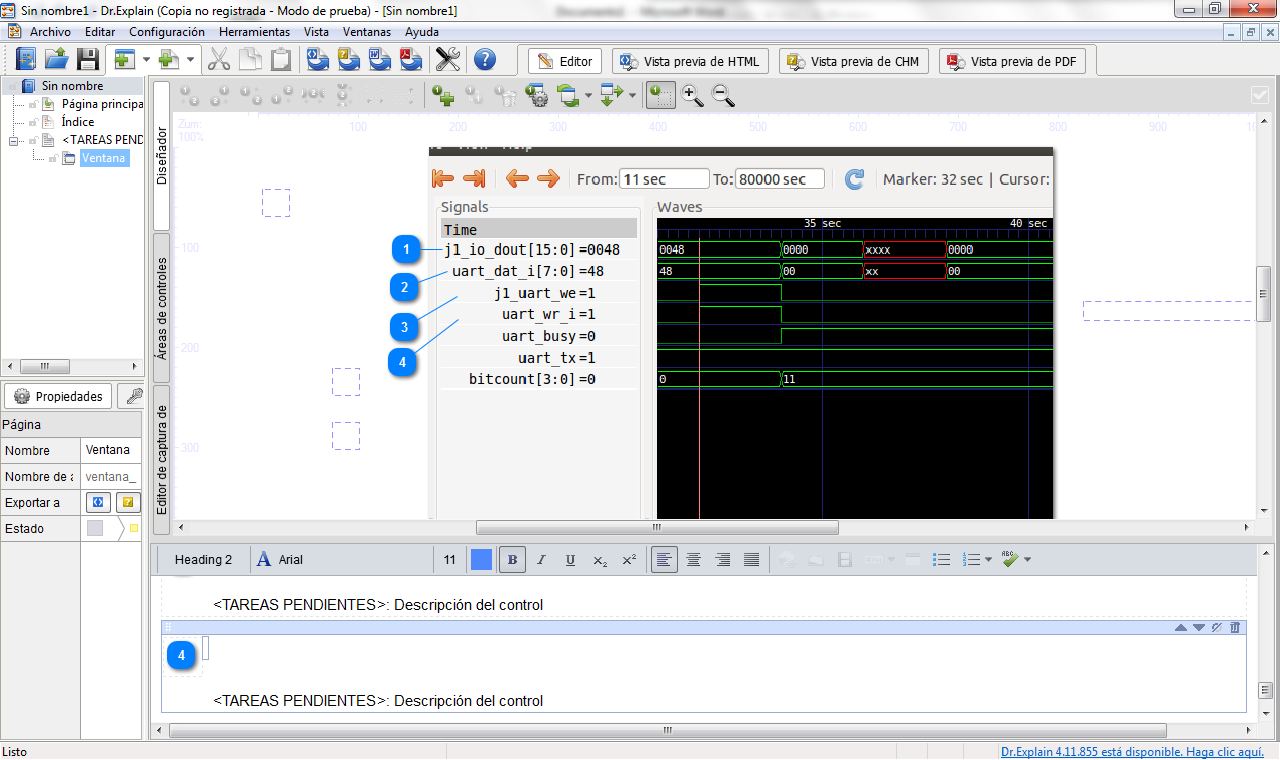


Fig.3.1 Señales de control CPU-UART

Los datos del mensaje en 8 bits entran desde la CPU al módulo UART a través de uart\_dat\_i (Fig.3.1-3.2). En el momento en el que la CPU desea realizar la transmisión, pone en alto la línea de j1\_uart\_we (Fig.1-3) la cual está conectada con uart\_wr\_i (Fig.-3.4), así el periférico sabrá que debe empezar la transmisión.

*Transmisión de datos*

Para la transmisión de datos se usa el protocolo estándar de 11 bits

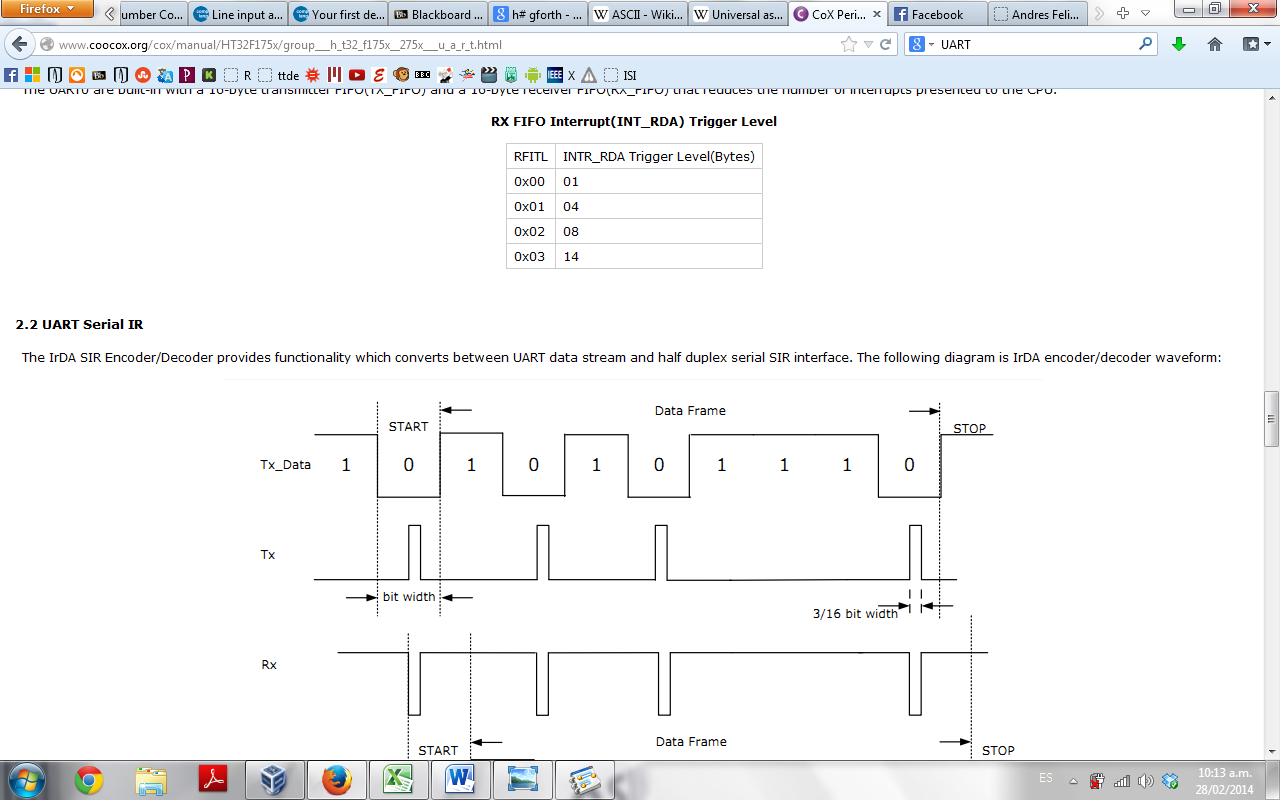


Fig.3.2 Protocolo de transmisión UART [1]

La línea de Tx permanece en alto antes de empezar la transmisión, luego se inicia con un bit de START (low), a continuación se encuentra el Frame del mensaje en este caso se utilizan 8 bits y finalmente se envía el bit de parada STOP (high).

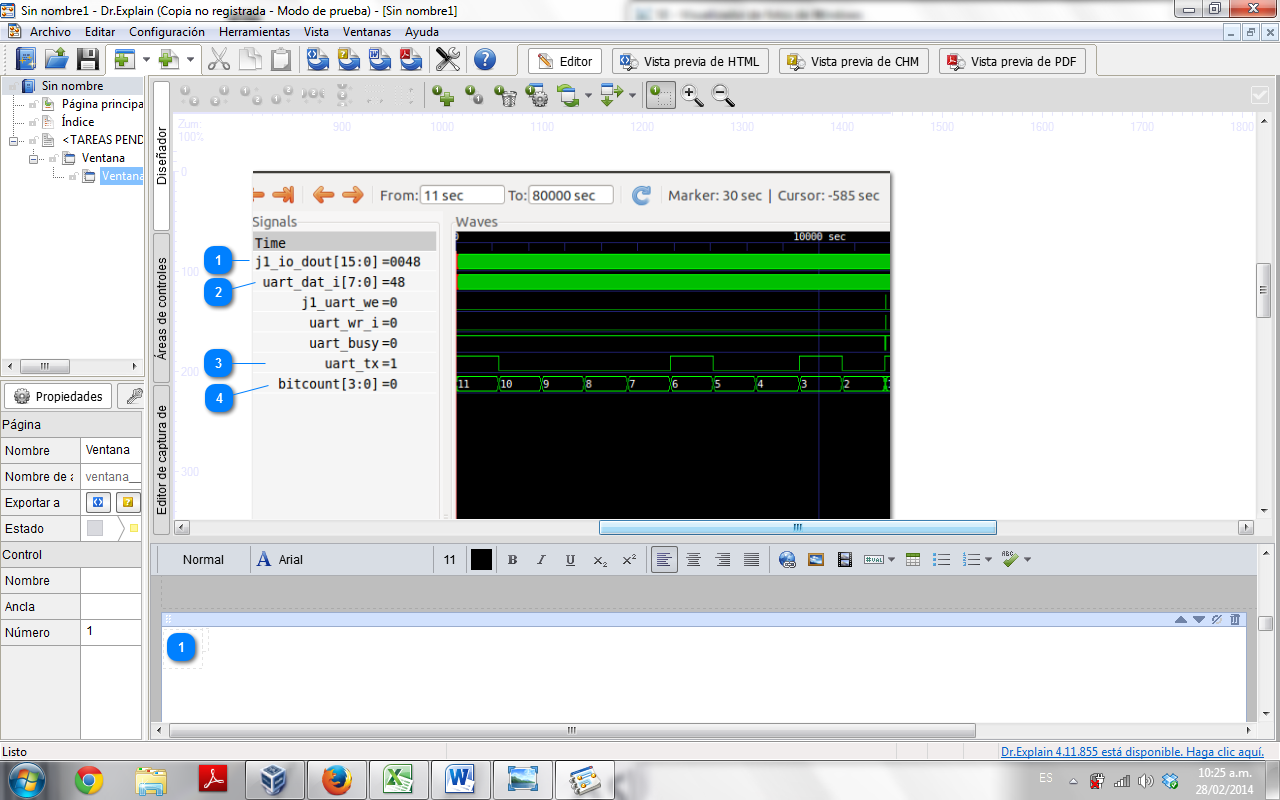


Fig.3.3 Transmisión de la letra H

La figura 3 muestra la transmisión del primer carácter “H”. Justo antes de habilitar el “write enable de la UART para la transmisión, la CPU tiene en la señal j1\_io\_dout (Fig.3.3-3.1) el carácter que desea trasmitir “H” (0x48) el cual, como anteriormente se explicó, está comunicado con uart\_dat\_i (Fig.3.2). El bloque UART concatena el frame del mensaje con los bits de START y STOP para completar el protocolo de 11 bits. El registro bitcount (Fig3.3.-3.4) permite ver la secuencia de transmisión desde el bit 11 hasta el 1. Luego de habilitar la UART y establecer el paquete de 11bits se empieza la transmisión por medio de uart\_tx (Fig.3.3). Como se aprecia en la figura 3, la transmisión por medio de uart\_x para este caso es: 10000100101. Donde (10) establece el inicio del mensaje START, (00010010) tienen el dato correspondiente a la letra “H” (0x48) transmitidos desde el LSB hasta el MSB. El bit uart\_busy en estado alto indica que el módulo tiene el canal ocupado y se encuentra transmitiendo.

De manera similar se inspecciona la simulación para comprobar el mensaje completo, la tabla 1 resume la secuencia del mensaje:

Tabla I. Secuencia del mensaje completo

|  |  |  |  |
| --- | --- | --- | --- |
| Caracter | uart\_tx | | |
| **START** | Data | **STOP** |
| H 0x48 | 10 | 00010010 | 1 |
| E 0x45 | 10 | 10100010 | 1 |
| L 0x4C | 10 | 00110010 | 1 |
| L 0x4C | 10 | 00110010 | 1 |
| O 0x4F | 10 | 11110010 | 1 |

# Ejemplo de programación serie de Fibonacci usando lenguaje GForth

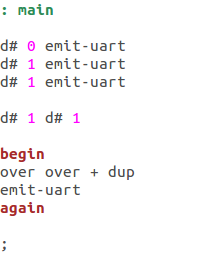


Fig.4 Programa de la aplicación principal

En la figura 4 se encuentra el programa que envía por UART la serie de Fibonacci. Los tres primeros valores de la serie: 0, 1, 1 se ingresan al Stack de memoria indicando, con el prefijo d#, que se trata de números en formato decimal.

Inmediatamente después de que se ingresa número por número al Stack, se usa la función emit-uart que se encuentra en nuc.fs para transmitir el dato. Al usar la función emit-uart, el dato transmitido sale del Stack, por esto es necesario volver a introducir los dos últimos números ( d# 1 d# 1).

Con el número 1 en T y N , la posición más alta y la siguiente en el Stack, se empieza el ciclo para generar la serie automáticamente de manera indefinida. Con las instrucciones over over + se obtiene el siguiente número de la serie, finalmente se duplica el dato en T para conservarlo en el Stack luego de la transmisión por UART. El manual completo del lenguaje GForth se encuentra en [2]. Para la primera iteración se tiene:

Tabla II. Algoritmo Fibonacci, secuencia en Stack

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Posición Stack | Instrucción | | | | | |
| Estado inicial | over | over | + | dup | emit-uart |
| T | 1 | 1 | 1 | 2 | 2 | 2 |
| N | 1 | 1 | 1 | 1 | 2 | 1 |
| 3 |  | 1 | 1 | 1 | 1 | 1 |
| 4 |  |  | 1 |  | 1 |  |

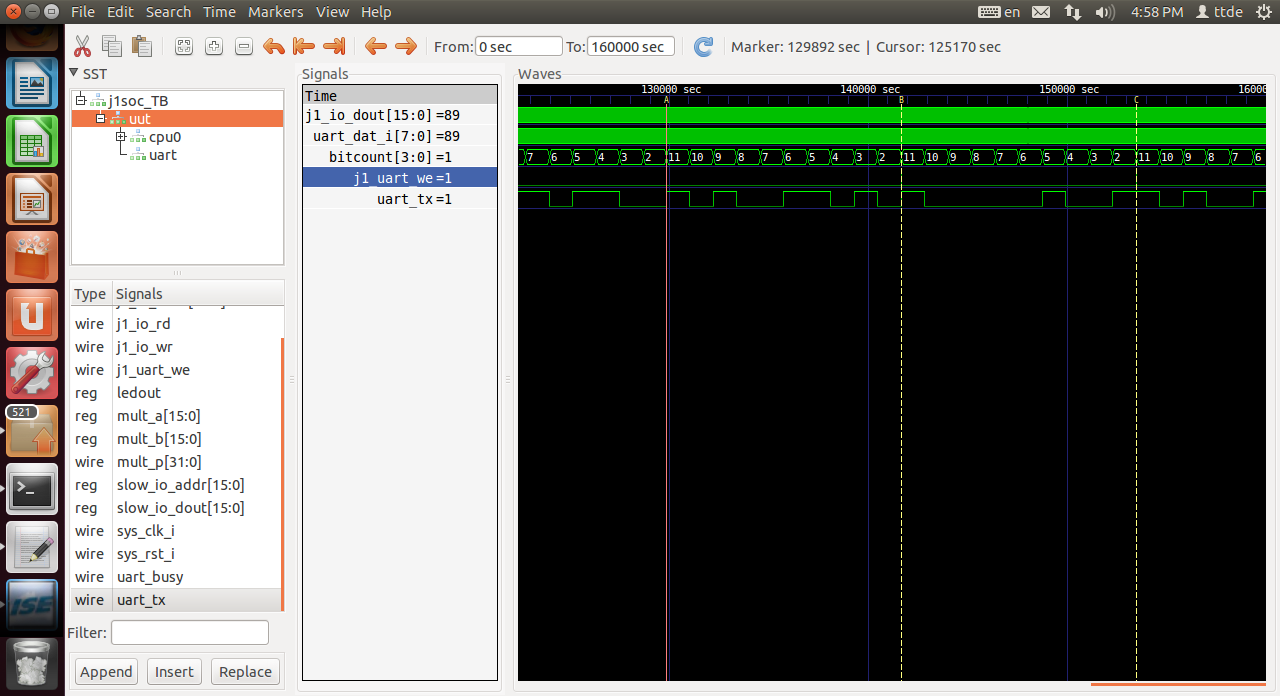
Segunda Iteración:

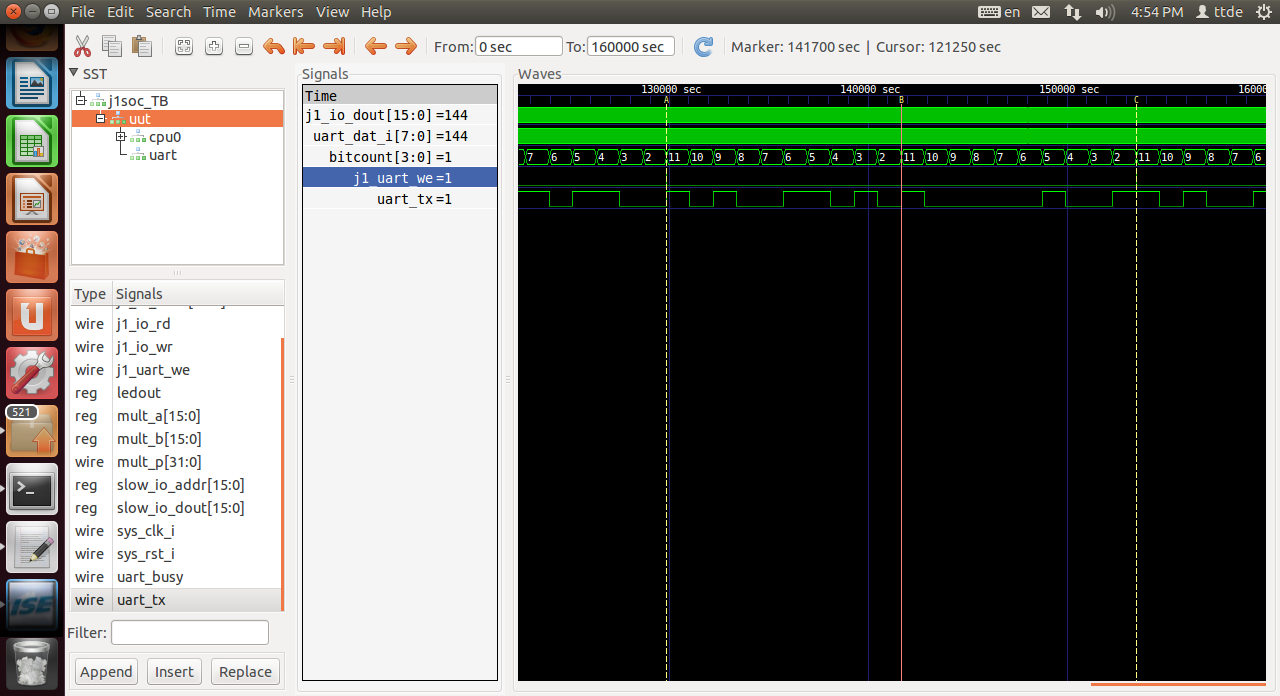
Tabla III. Algoritmo Fibonacci, secuencia en Stack segunda iteración

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Posición Stack | Instrucción | | | | | |
| Estado inicial | over | over | + | dup | emit-uart |
| T | 2 | 1 | 2 | 3 | 3 | 3 |
| N | 1 | 2 | 1 | 2 | 3 | 2 |
| 3 | 1 | 1 | 2 | 1 | 2 | 1 |
| 4 |  | 1 | 1 | 1 | 1 | 1 |
| 5 |  |  | 1 |  | 1 |  |

*Simulaciones*

Para realizar las simulaciones, se cambió el tiempo de simulación a 16s en el archivo j1\_soc\_TB.v, así es posible observar más iteraciones del algoritmo. La figura 5 muestra la simulación desde 13s a 16s en dónde se ve la generación de los números 12, 13 y 14 de la serie, es decir 89 (Fig. 5-A), (Fig.5-B) 144 y(Fig.5-C) 233.

A) Número 89 Serie de Fibonacci

B) Número 144 serie de Fibonacci

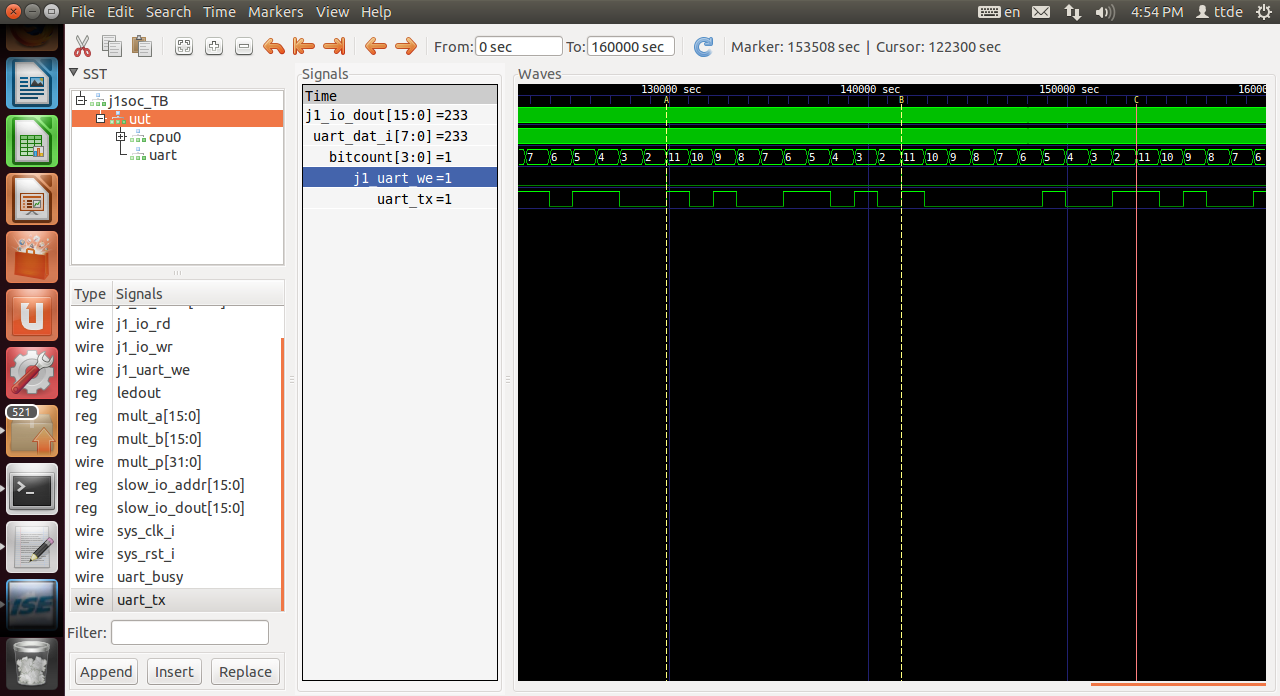
C) Número 233 Serie de Fibonacci

Fig.3.5 Simulaciones serie de Fibonacci.

Como se explicó en la sección anterior, la señal j1\_uart\_we, habilita al periférico UART para empezar la transmisión. En este momento el dato presente en uart\_dat\_i es trasmitido usando la señal uart\_tx.

**Referencias**

[1] CooCox, HT32F175x\_275x UART, Documento Online disponible en www.coocox.org/cox/manual/HT32F175x/group\_\_\_h\_t32\_f175x\_\_275x\_\_\_u\_a\_r\_t.html

[2] Complang, GForth Manual, Disponible en: www.complang.tuwien.ac.at/forth/gforth/Docs-html/

[3] J. Bowman. *J1 a small Forth CPU Core for FPGAs* Disponible en: http://excamera.com/files/j1.pdf